LAPORAN PRAKTIKUM DATA STRUCTURE

BINARY TREES

Dosen Pengampu:

H. Fatchurrochman,M.Kom

Asisten Praktikum:

Fillah Anjany 230605110033

Oleh :

Muhammad Alif Mujaddid **240605110082**

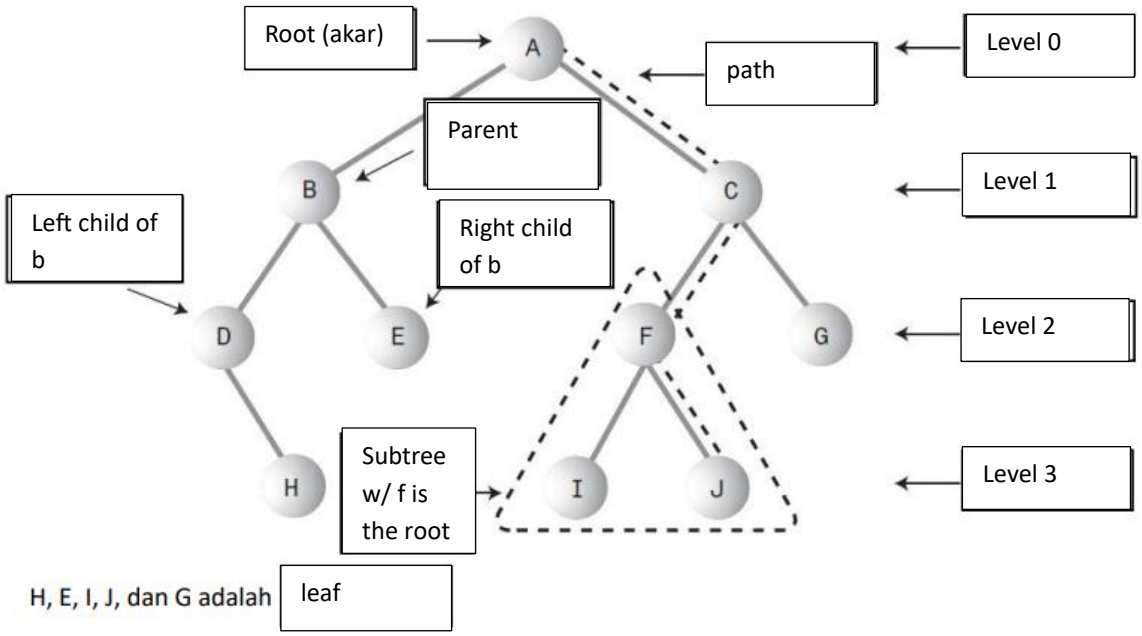
Jurusan Teknik Informatika Fakultas Sains dan Teknologi

UIN Maulana Malik Ibrahim Malang

2025

A. PENDAHULUAN

1. Untuk mengenali istilah-istilah yang ada pada tree, isilah tiap kotak pada gambar berikut dengan istilah yang sesuai.



2. Jelaskan apa yang dimaksud dengan path, root, parent, child, leaf, subtree, visiting, traversing, level, dan key pada tree.

Path :	Jalur yang menghubungkan satu node ke node lainnya melalui serangkaian edge (garis).
Root :	Node paling atas dari pohon, tidak memiliki parent.
Parent :	Node yang memiliki satu atau lebih child (anak).
Child :	Node yang memiliki parent di atasnya.
Leaf :	Node yang tidak memiliki child (simpul daun).
Subtree :	Bagian pohon yang terdiri dari sebuah node dan seluruh keturunannya.
Visiting :	Proses “mengunjungi” sebuah node (misalnya saat traversal).
Traversing :	Proses mengunjungi setiap node dalam pohon dengan urutan tertentu (preorder, inorder, postorder).
Level :	Menunjukkan kedalaman node dari root (root = level 0 atau 1).
Key :	Nilai atau data yang disimpan di dalam node.

3. Implementasi Binary tree pada Java Tree dapat diimplementasikan dengan menyimpan node pada memori sebagai sebuah array. Atau, tree juga dapat diimplementasikan dengan menyimpan node pada memori dan menghubungkannya menggunakan references pada tiap node yang menunjuk pada children node

tersebut. Berikut ini beberapa class yang perlu implementasikan dalam membuat program binary tree.

a. Class Node Class Node sebagai objek yang merepresentasikan data objek yang akan disimpan dan juga memuat reference ke masing-masing dua children
b. Class Tree Class Tree merepresentasikan objek tree itu sendiri, yaitu sebuah objek yang menangani semua node. Pada class Tree, hanya ada 1 field, yaitu variable node dengan nama root. Node lain diakses melalui root sehingga tidak diperlukan adanya field lain. Terdapat beberapa method pada class Tree, yaitu method untuk pencarian (find), menambahkan node (insert), menghapus node (delete), serta method yang menangani berbagai macam travers, dan untuk menampilkan tree (display).
c. Class TreeApp Class TreeApp berisi method main untuk menjalankan program. Pada listing program berikut, method main menyediakan sebuah primitive user interface sehingga user dapat menentukan aksi dari input keyboard, apakah ingin insert, find, delete, show, atau travers.
   Tuliskan listing program berikut ini. Lengkapi dengan class Stack yang digunakan pada method display di class Tree.

TreeApp.java Praktikum\modul7 9+, U

```java
package Praktikum.modul7;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Stack;

class Node {
    public int id;
    public String data;
    public Node leftChild;
    public Node rightChild;

    public void displayNode() {
        System.out.print("{ " + id + ", " + data + " } ");
    }
}
```

```
19  class Tree {
20      private Node root;
21      public Tree() {
22          root = null;
23      }
24
25      public Node find(int key) {
26          Node current = root;
27          while (current.id ≠ key) {
28              if (key < current.id) {
29                  current = current.leftChild;
30              } else {
31                  current = current.rightChild;
32              }
33              if (current == null) {
34                  return null;
35              }
36          }
37          return current;
38      }
```

```java
public void insert(int id, String data) {
    Node newNode = new Node();
    newNode.id = id;
    newNode.data = data;
    if (root == null) {
        root = newNode;
    } else {
        Node current = root;
        Node parent;
        while (true) {
            parent = current;
            if (id < current.id) {
                current = current.leftChild;
                if (current == null) {
                    parent.leftChild = newNode;
                    return;
                }
            } else {
                current = current.rightChild;
                if (current == null) {
                    parent.rightChild = newNode;
                    return;
                }
            }
        }
    }
}
```

```
96                  root = current.leftChild;
97              } else if (isLeftChild) {
98                  parent.leftChild = current.leftChild;
99              } else {
100                 parent.rightChild = current.leftChild;
101             }
102         } else if (current.leftChild == null) {
103             if (current == root) {
104                 root = current.rightChild;
105             } else if (isLeftChild) {
106                 parent.leftChild = current.rightChild;
107             } else {
108                 parent.rightChild = current.rightChild;
109             }
110         } else {
111             Node successor = getSuccessor(current);
112             if (current == root) {
113                 root = successor;
114             } else if (isLeftChild) {
115                 parent.leftChild = successor;
116             } else {
117                 parent.rightChild = successor;
118             }
119             successor.leftChild = current.leftChild;
120         }
121         return true;
122     }
```

```java
    private Node getSuccessor(Node delNode) {
        Node successorParent = delNode;
        Node successor = delNode;
        Node current = delNode.rightChild;
        while (current != null) {
            successorParent = successor;
            successor = current;
            current = current.leftChild;
        }
        if (successor != delNode.rightChild) {
            successorParent.leftChild = successor.rightChild;
            successor.rightChild = delNode.rightChild;
        }
        return successor;
    }
```

```java
    public void traverse(int traverseType) {
        switch (traverseType) {
            case 1:
                System.out.print(s:"Preorder traversal: ");
                preOrder(root);
                break;
            case 2:
                System.out.print(s:"Inorder traversal: ");
                inOrder(root);
                break;
            case 3:
                System.out.print(s:"Postorder traversal: ");
                postOrder(root);
                break;
        }
        System.out.println();
    }

    private void preOrder(Node localRoot) {
        if (localRoot ≠ null) {
            System.out.print(localRoot.id + " ");
            preOrder(localRoot.leftChild);
            preOrder(localRoot.rightChild);
        }
    }
```

```
166     private void inOrder(Node localRoot) {
167         if (localRoot != null) {
168             inOrder(localRoot.leftChild);
169             System.out.print(localRoot.id + " ");
170             inOrder(localRoot.rightChild);
171         }
172     }
173
174     private void postOrder(Node localRoot) {
175         if (localRoot != null) {
176             postOrder(localRoot.leftChild);
177             postOrder(localRoot.rightChild);
178             System.out.print(localRoot.id + " ");
179         }
180     }
```

```
 19  class Tree {
182      public void displayTree() {
183          Stack globalStack = new Stack();
184          globalStack.push(root);
185          int nBlanks = 32;
186          boolean isRowEmpty = false;
187          System.out.println(
188                  x:"........................................");
189          while (isRowEmpty == false) {
190              Stack localStack = new Stack();
191              isRowEmpty = true;
192              for (int j = 0; j < nBlanks; j++) {
193                  System.out.print(c:' ');
194              }
195              while (globalStack.isEmpty() == false) {
196                  Node temp = (Node) globalStack.pop();
197                  if (temp != null) {
198                      System.out.print(temp.id);
199                      localStack.push(temp.leftChild);
200                      localStack.push(temp.rightChild);
201                      if (temp.leftChild != null
202                              || temp.rightChild != null) {
203                          isRowEmpty = false;
204                      }
205                  } else {
206                      System.out.print(s:"--");
207                      localStack.push(item:null);
208                      localStack.push(item:null);
209                  }
```

```
19  class Tree {
182     public void displayTree() {
210             for (int j = 0; j < nBlanks * 2 - 2; j++) {
211                 System.out.print(c:' ');
212             }
213         }
214         System.out.println();
215         nBlanks /= 2;
216         while (localStack.isEmpty() == false) {
217             globalStack.push(localStack.pop());
218         }
219     }
220     System.out.println(
221             x:"..........................................");
222 }
223 }
224
225 public class TreeApp {
        Run | Debug
226     public static void main(String[] args) throws IOException {
227         int value;
228         String data;
229         Tree theTree = new Tree();
230         theTree.insert(id:50, data:"Ahmad");
231         theTree.insert(id:25, data:"Rosa");
232         theTree.insert(id:75, data:"Raisa");
233         theTree.insert(id:12, data:"Naya");
234         theTree.insert(id:37, data:"Gagas");
235         theTree.insert(id:43, data:"Ainun");
236         theTree.insert(id:30, data:"Beri");
237         theTree.insert(id:33, data:"Vivid");
238         theTree.insert(id:87, data:"Orin");
239         theTree.insert(id:93, data:"Wiwid");
240         theTree.insert(id:97, data:"Sasa");
241         while (true) {
242             System.out.print("Enter first letter of show, "
```

```
241         while (true) {
242             System.out.print("Enter first letter of show, "
243                     + "insert, find, delete, or traverse: ");
244             int choice = getChar();
245             switch (choice) {
246                 case 's':
247                     theTree.displayTree();
248                     break;
249                 case 'i':
250                     System.out.print("Enter value and data to"
251                             + " insert: ");
252                     value = getInt();
253                     data = getString();
254                     theTree.insert(value, data);
255                     break;
256                 case 'f':
257                     System.out.print(s:"Enter value to find: ");
258                     value = getInt();
259                     Node found = theTree.find(value);
260                     if (found ≠ null) {
261                         System.out.print(s:"Found: ");
262                         found.displayNode();
263                         System.out.print(s:"\n");
264                     } else {
265                         System.out.println("Could not find "
266                                 + value);
267                     }
268                     break;
```

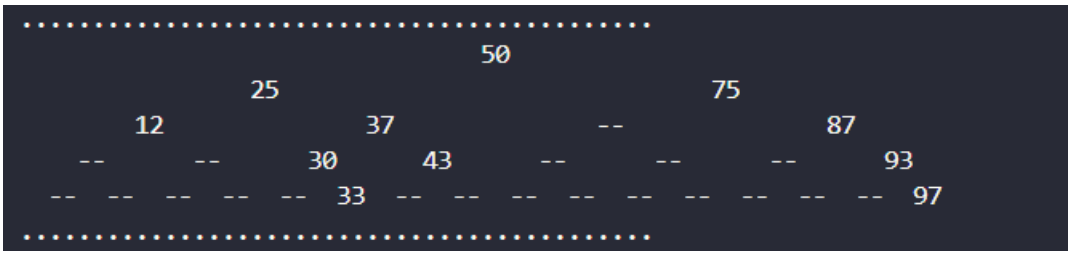
```java
public class TreeApp {
    public static void main(String[] args) throws IOException {
                case 'd':
                    System.out.print(s:"Enter value to delete: ");
                    value = getInt();
                    boolean didDelete = theTree.delete(value);
                    if (didDelete) {
                        System.out.println("Deleted " + value);
                    } else {
                        System.out.println("Could not delete "
                                + value);
                    }
                    break;
                case 't':
                    System.out.print(s:"Enter type 1, 2 or 3: ");
                    value = getInt();
                    theTree.traverse(value);
                    break;
                default:
                    System.out.println(x:"Invalid entry ");
            }
        }
    }

    public static String getString() throws IOException {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        String s = br.readLine();
        return s;
    }

    public static char getChar() throws IOException {
        String s = getString();
        return s.charAt(index:0);
    }
```

```java
    public static String getString() throws IOException {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        String s = br.readLine();
        return s;
    }

    public static char getChar() throws IOException {
        String s = getString();
        return s.charAt(index:0);
    }

    public static int getInt() throws IOException {
        String s = getString();
        return Integer.parseInt(s);
    }
}
```

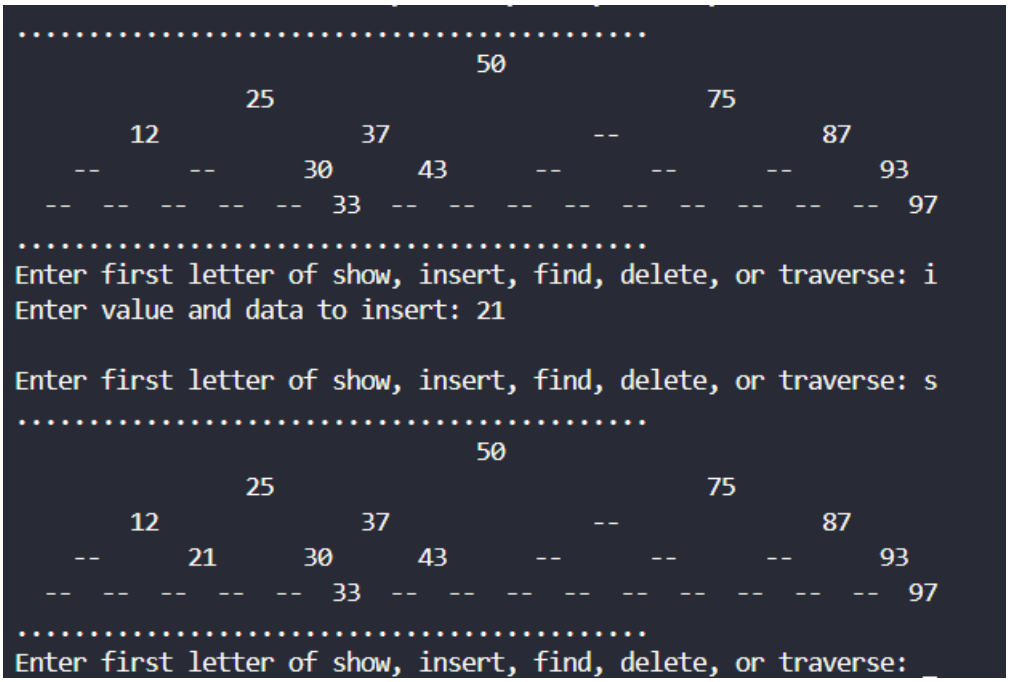
```
Enter first letter of show, insert, find, delete, or traverse: s
.........................................
                          50
              25                          75
        12            37            --            87
    --      --      30      43      --      --      --      93
  --  --  --  --  --  33  --  --  --  --  --  --  --  --  --  97
.........................................
Enter first letter of show, insert, find, delete, or traverse:
```

```
addid@LAPTOP-F80MDN28 MINGW64 /a/Code/DSA (main)
$  cd a:\\Code\\DSA ; /usr/bin/env A:\\Java\\jdk-22\\bin\\java.exe -agentlib:j
:\\Users\\addid\\AppData\\Roaming\\Code\\User\\workspaceStorage\\df75c84a286ea
Enter first letter of show, insert, find, delete, or traverse: i
Enter value and data to insert: 10
addid
Enter first letter of show, insert, find, delete, or traverse: s
.........................................
                          50
              25                          75
        12            37            --            87
    10      --      30      43      --      --      --      93
  --  --  --  --  --  33  --  --  --  --  --  --  --  --  --  97
.........................................
Enter first letter of show, insert, find, delete, or traverse:
```

4. Tuliskan output program pada listing nomor 3 dalam bentuk tree ketika memilih menu Show

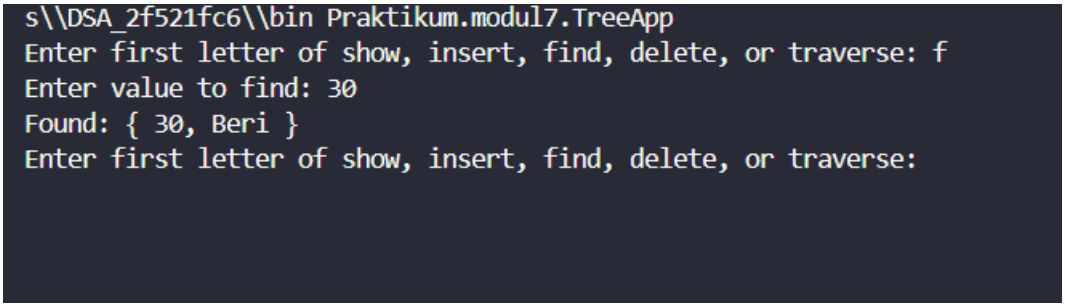


5. Tambahkan satu node dengan value tertentu! Tampilkan tree, tunjukkan dimana posisi node yang baru ditambahkan, dan jelaskan langkah penambahan node yang diimplementasikan pada method insert!



Node 21 dimasukkan sebagai anak kiri dari node 25 setelah node 12, karena:
21 < 50, turun ke kiri (ke 25)
21 < 25, turun ke kiri (ke 12)
21 > 12, ditempatkan sebagai anak kanan dari node 12

6. Lakukan percarian salah satu leaf pada tree tersebut! Tuliskan outputnya dan penjelasan langkah pencarian yang diimplementasikan pada method find! Anda dapat menggambarkan ilustrasi pencarian node tersebut pada tree sebagai visualisasi langkah pencarian.



Proses pencarian node dengan nilai 30 dilakukan menggunakan method find(int key). Pencarian dimulai dari root tree, yaitu node dengan nilai 50. Karena nilai yang

dicari (30) lebih kecil dari 50, maka pencarian dilanjutkan ke anak kiri dari node tersebut, yaitu node 25.
Selanjutnya, pada node 25, dibandingkan kembali bahwa 30 lebih besar dari 25, sehingga pencarian berpindah ke anak kanan dari node 25, yaitu node 37. Di node 37, nilai 30 lebih kecil dari 37, maka pencarian bergerak ke anak kiri dari node tersebut. Pada langkah ini, ditemukan node dengan nilai 30 yang sesuai dengan nilai yang dicari.

7. Proses hapus node pada tree cukup kompleks dibandingkan dengan operasi lain. Langkah pertama yang dilakukan adalah mencari node yang akan dihapus. Terdapat 3 kemungkinan kondisi node yang akan dihapus, yaitu:
   a. Node yang akan dihapus adalah leaf, yang berarti node tersebut tidak memiliki children. Untuk kondisi ini, langkah penghapusan cukup sederhana
   b. Node yang akan dihapus memiliki satu child. Penghapusan node pada kondisi ini tidak semudah kondisi pertama
   c. Node yang akan dihapus memiliki dua children. Penghapusan node kondisi ini cukup rumit. Pada program nomor 3, hapuslah 3 node yang mewakili masing-masing kemungkinan kondisi node diatas. Gambarkan ilustrasi proses hapus masing-masing node tersebut berdasarkan langkah yang diimplentasikan pada program nomor 3.

8. Terdapat tiga jenis operasi traverse, yaitu preorder, inorder, dan postorder. pada program nomor 3, jalankan masing-masing operasi traverse tersebut, tulis outputnya dan jelaskan perbedaan ketiga operasi tersebut!

```
Enter first letter of show, insert, find, delete, or traverse: t
Enter type 1, 2 or 3: 1
Preorder traversal: 50 25 12 37 30 33 43 75 87 93 97
Enter first letter of show, insert, find, delete, or traverse: t
Enter type 1, 2 or 3: 2
Inorder traversal: 12 25 30 33 37 43 50 75 87 93 97
Enter first letter of show, insert, find, delete, or traverse: t
Enter type 1, 2 or 3: 3
Postorder traversal: 12 33 30 43 37 25 97 93 87 75 50
Enter first letter of show, insert, find, delete, or traverse: _
```